

Sablier Airdrops

Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher
Rustyrabbit, Security Researcher

July 28, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Anyone can prematurely force-claim MerkleVCA airdrops for all recipients causing forfeiture of significant airdrop funds . . . 4
- 3.2 Medium Risk . . . 4
  - 3.2.1 Counterfactual deployments of `SablierMerkleVCA` may later revert to cause loss of funds . . . 4
- 3.3 Low Risk . . . 5
  - 3.3.1 Static `DOMAIN_SEPARATOR` will cause `claimViaSig()` signature failure on a chain split . . . 5
  - 3.3.2 Missing `validFrom` in signatures may lead to loss of funds for relayed `MerkleVCA` airdrop claims . . . 6
  - 3.3.3 User funds can be accidentally withdrawn as fees if `setNativeToken()` is not called timely . . . 7
- 3.4 Gas Optimization . . . 7
  - 3.4.1 `unlockAmount` calculation can be delayed . . . 7
- 3.5 Informational . . . 7
  - 3.5.1 Malicious/compromised airdrop admins can abuse clawback functionality causing loss of funds to eligible participants . . . 7
  - 3.5.2 Malicious airdrop campaign creator can grief claimants by insufficiently funding campaign . . . 8
  - 3.5.3 Comptroller's ability to only lower the fee can be circumvented by setting a malicious oracle . . . 9
  - 3.5.4 Missing `ComptrollerManager` convenience function to query required fee . . . 9
  - 3.5.5 Returning zero in `calculateForgoneAmount()` when `claimTime < VESTING_START_TIME` may be unexpected . . . 9
  - 3.5.6 Forcing `MerkleVCA` campaigns to expire only after a week of `vestingEndTime` may be too restrained . . . 10

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Sablier Protocol is a collection of persistent, non-upgradeable smart contracts to facilitate streaming of ERC-20 tokens on Ethereum and other EVM blockchains. It consists of Lockup, Merkle Airdrops, and Flow.

From Jun 19th to Jul 3rd the Cantina team conducted a review of airdrops on commit hash 5088c4ce. All contracts under `src` (except `interfaces`) are in scope. The team identified a total of **12** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	3	2	1
Gas Optimizations	1	1	0
Informational	6	2	4
Total	**12**	**7**	**5**

The Cantina Managed team reviewed Sablier's `airdrops` holistically on commit hash d4eb8def and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 High Risk

3.1.1 Anyone can prematurely force-claim MerkleVCA airdrops for all recipients causing forfeiture of significant airdrop funds

Severity: High Risk

Context: SablierMerkleVCA.sol#L118-L143, SablierMerkleVCA.sol#L150-L165

Summary: Anyone can prematurely force-claim MerkleVCA airdrops via `claim()` for all recipients causing forfeiture of significant airdrop funds for them.

Finding Description: `MerkleVCA` supports the Variable Claim Amount (VCA) airdrop model, where there is an initial `UNLOCK_PERCENTAGE` amount (percentage of the full amount that will unlock immediately at the start time, e.g. can be as low as 1%) after which the claim amount increases linearly until the airdrop period ends. Claiming early results in forgoing the remaining amount, whereas claiming after the vesting period grants the full amount that was allocated. This is captured in the below logic:

```
function _calculateClaimAmount(uint128 fullAmount, uint40 claimTime) private view returns (uint128) {
    // If the claim time is less than the vesting start time, there's nothing to calculate, so we return zero.
    if (claimTime < VESTING_START_TIME) {
        return 0;
    }

    // Calculate the initial unlock amount.
    uint128 unlockAmount = ud(fullAmount).mul(UNLOCK_PERCENTAGE).intoUint128();

    // If the vesting period has ended, the full amount can be claimed.
    if (claimTime >= VESTING_END_TIME) {
        return fullAmount;
    }
    // Otherwise, calculate the claim amount based on the elapsed time.
    else {
        uint40 elapsedTime;
        uint40 totalDuration;

        // Safe because overflows are prevented by the checks above.
        unchecked {
            elapsedTime = claimTime - VESTING_START_TIME;
            totalDuration = VESTING_END_TIME - VESTING_START_TIME;
        }

        // Safe to cast because the result is less than `remainderAmount`, which fits within `uint128`.
        uint256 remainderAmount = uint256(fullAmount - unlockAmount);
        uint128 vestedAmount = uint128((remainderAmount * elapsedTime) / totalDuration);
        return unlockAmount + vestedAmount;
    }
}
```

However, `MerkleVCA` implements an external `claim()` function that can be called by anyone for any recipient at any time. This allows anyone to trigger the airdrop claim for all recipients as soon as the vesting time starts, which will force claim only the `UNLOCK_PERCENTAGE` of the airdrop claim amount and forfeit the remaining (presumably significant) percentage of funds. Given that this airdrop model is meant to incentivize recipients to claim as late as possible (preferably after `VESTING_END_TIME`), the `UNLOCK_PERCENTAGE` is typically expected to be small.

Project communicated that they independently discussed this issue during the review period.

Impact Explanation: High, because a significant percentage of the airdrop claimable funds can be force-forfeited for all recipients by anyone at anytime. This can result in significant loss of funds for all airdrop recipients. This may also negatively affect the airdrop project's reputation.

Likelihood Explanation: Medium, because the attacker may not always benefit by forced forfeiture besides griefing. However, if the airdrop campaign is designed to redistribute forfeited funds to those claiming later, then malicious participants themselves can force-claim others while hoping to not get force-claimed themselves, which gives them a financial incentive to exploit this vector. The campaign creator themself may indirectly trigger such premature claims to later clawback the forfeited tokens after campaign ends.

Recommendation: Consider removing the `claim()` functionality for `MerkleVCA` where it can be abused unlike the other airdrop models of `MerkleInstant`, `MerkleLL`, and `MerkleLT`.

Sablier: Fixed in PR 173.

Cantina Managed: Fix verified. PR 173 removes `claim()`.

3.2 Medium Risk

3.2.1 Counterfactual deployments of `SablierMerkleVCA` may later revert to cause loss of funds

Severity: Medium Risk

Context: SablierFactoryMerkleVCA.sol#L43-L62, SablierMerkleVCA.sol#L60-L106

Summary: `createMerkleVCA()` can revert during parameter sanity checks to cause loss of funds for counterfactual deployments.

Finding Description: Merkle airdrop contracts are deployed deterministically using CREATE2 in their respective factories. This allows counterfactual deployment (a technique where a smart contract's address is precomputed and can be used before the contract is actually deployed on the blockchain). So in the case of Sablier Merkle airdrop contracts, the airdrop campaign creator can determine their airdrop contract's address using `ConstructorParams param` values they plan to use and fund that address with the airdrop amount, all before deployment.

However, if the actual deployment of such a contract fails due to a revert in any of the sanity checks for their address-determining parameters within their constructor, any funds deposited to the counterfactually deployed contract earlier is permanently lost because fixing the revert causing parameter values will also change the deployment address. Contracts deployed using CREATE2 therefore typically do not perform any reverting checks in their constructors.

While the constructors of `SablierMerkleInstant`, `SablierMerkleLL`, and `SablierMerkleLT` avoid such reverting checks to only perform them later during claims, `SablierMerkleVCA` introduces them back for `params.vestingStartTime == 0`, `params.vestingEndTime <= params.vestingStartTime`, `params.expiration == 0`, `params.expiration < params.vestingEndTime + 1 weeks` and `params.unlockPercentage.unwrap() > uUNIT`. The motivation for this is to identify any incorrect deployment parameters before claims, which are incentivized in this airdrop model to happen only closer/beyond the vesting period unlike in the other variants where they may happen much earlier.

A similar issue was reported in an earlier review (see finding 3.1.8).

Impact Explanation: High, because any funds deposited to the counterfactually deployed `SablierMerkleVCA` contract is permanently lost, which could cause a significant fund loss to the campaign creator.

Likelihood Explanation: Low, because this requires a campaign creator to counterfactually deposit airdrop funds and also have reverting errors in their supplied parameters during deployment.

Recommendation: Consider:

1. Moving all the reverting checks from the constructor to `_postProcessClaim()` as done with other airdrop variants.
2. To detect any deployment parameter errors early, consider adding a dummy recipient (controlled by the campaign) to the airdrop claimants with a trivial amount (e.g. 1 wei), which can be claimed soon after deployment and also trigger any errors if present. As documented in an earlier report, this will also help trigger the claim period to prevent admin abuse of the campaign functionality.
3. Consider providing a view function to generate the deployment address using the same reverting checks that will help catch incorrect deployment parameters before the actual deployment.

Sablier: Fixed in PR 178.

Cantina Managed: Reviewed that PR 178 mitigates the issue as recommended in (3) to:

1. Provide a `computeMerkleVCA()` function, which implements all the checks in `_checkDeploymentParams()` that is also called in `createMerkleVCA()`. All checks are removed from constructor and `_postProcessClaim()` . Creator is expected to call `computeMerkleVCA()` beforehand to ensure parameter correctness for any counterfactual funding. If they still accidentally use incorrect

parameters for a contract that is counterfactually funded, then they can deploy contract themselves (bypassing the create factory function) to recover any stuck funds from incorrect parameters.

2. Adds a similar `compute*()` and function `create*()` for other airdrop types.

3.3 Low Risk

3.3.1 Static `DOMAIN_SEPARATOR` will cause `claimViaSig()` signature failure on a chain split

Severity: Low Risk

Context: SablierMerkleBase.sol#L82-L98, SablierMerkleBase.sol#L173-L194, SablierMerkleVCA.sol#L187-L201

Summary: Static `DOMAIN_SEPARATOR` initialized during Merkle airdrop contract deployments will cause signature failure on a chain split.

Finding Description: Merkle airdrop contracts statically initialize an immutable `DOMAIN_SEPARATOR` in the constructor where `block.chainid` and `address(this)` are read once during contract deployments as shown below:

```
// Compute the domain separator to be used for claiming using an EIP-712 or EIP-1271 signature.
DOMAIN_SEPARATOR = keccak256(
    abi.encode(SignatureHash.DOMAIN_TYPEHASH, SignatureHash.PROTOCOL_NAME, block.chainid, address(this))
);
```

This is used by `_checkSignature()` in `claimViaSig()` to create the digest `bytes32 digest = MessageHashUtils.toTypedDataHash({ domainSeparator: DOMAIN_SEPARATOR, structHash: claimHash });` used in `SignatureChecker.isValidSignatureNow({ signer: recipient, hash: digest, signature: signature })`.

However, such a static initialization, which does not dynamically check a cached version of contract address and chain ID to rebuild the separator if necessary, does not account for any chain split resulting in a new chain ID (or a redeployed contract at a different address). Such scenarios will result in signature verification failures.

Impact Explanation: Low, because this will cause prior signatures to become invalid in the event of any hard fork resulting in a new chain ID.

Likelihood Explanation: Very Low, because this issue only arises in the event of any hard fork resulting in a new chain ID.

Recommendation: Consider using OpenZeppelin EIP712 or its approach to caching and dynamically reusing/rebuilding the domain separator as necessary.

Sablier: Fixed in PR 176.

Cantina Managed: Reviewed that PR 176 mitigates the issue by caching chain ID and dynamically checking and rebuilding domain separator as necessary, similar to Uniswap Permit2

3.3.2 Missing `validFrom` in signatures may lead to loss of funds for relayed `MerkleVCA` airdrop claims

Severity: Low Risk

Context: SablierMerkleBase.sol#L173-L194, SablierMerkleVCA.sol#L187-L208

Summary: Missing `validFrom` in signatures may lead to loss of funds for `MerkleVCA` airdrop claims via `claimViaSig()`.

Finding Description: Claiming airdrops via signatures does not provide the time restrains and ability to revoke typically associated with offchain signatures. `claimViaSign()` provides a way for users to claim their airdrop via an offchain signature. This signature can be then transmitted by a relayer that pays for the gas and the protocol fee.

However, offchain signatures typically have protections where their validity is restricted in time and the user always have the possibility to revoke the signature onchain before they get included in cases when they no longer wish the signature to be executed. The current implementation does not provide these features. Doing so would increase flexibility of the claim via signature flows.

Also, once the signature is provided to a relayer service even with time restriction in place a user may change their mind and want to revoke the signature before it gets executed on chain. In order to do so a nonce would be required so that the user can increase the onchain nonce reference thereby invalidating any signatures the use the old nonce.

Impact Explanation: Medium, because a relayer for VCA airdrops can execute `claimViaSign()` anytime which may lead to loss of forfeited funds for the participant if it is before the vesting end time. The claimant has no onchain control over when `claimViaSig()` is triggered by the relayer to minimize forfeited amount unless they provide a valid signature only after such a time.

Likelihood Explanation: Low, if we assume that a VCA airdrop participant only relays their signature when they are ready to claim. If not, the relayer has no information on the intended claim time to maximize claimed amount and may claim anytime thereafter.

Recommendation: Consider:

1. Implementing `validFrom` at least for MerkleVCA airdrops in the signature.
2. Implementing `validTo` and `nonce` fields in the signature along with a function to invalidate old nonces on chain.

Sablier: It makes sense to add `validFrom` which could benefit users, particularly in case of VCA campaign. Regarding `validTo` and `nonce`, we've decided not to implement them because we don't think there could be a compelling use case for these features. Users generally want to claim their airdrops always, and therefore it is highly unlikely that someone would want to revoke their signature. If they accidentally create a wrong signature, they can use `claimTo` or `claim` to claim their airdrop and essentially revoking the signature validity. See PR 177.

Cantina Managed: Reviewed that PR 177 mitigates the issue as recommended in (1) for all airdrops.

3.3.3 User funds can be accidentally withdrawn as fees if `setNativeToken()` is not called timely

Severity: Low Risk

Context: SablierFlow.sol#L458-L474

Summary: `setNativeToken` should be called as soon as possible either immediately at deployment or whenever the native ERC20 contract is created (e.g. a network upgrade). Otherwise, a user can create streams with the native token and user funds can then be accidentally withdrawn as fees.

Finding Description: When a chain has an ERC20 representation for its native token `setNativeToken()` allows the controller to set it as native token so that no one can create streams with that token, because doing so would mean that fee withdrawal would end up withdrawing user funds instead of only the fees.

The native token is not set in the constructor and is meant to be called during deployment (although the provided scripts do not do so). The function is also necessary when a chain would implement the ERC20 representation of its native token in a fork or upgrade.

However, if anyone creates a stream with the token between the deployment of the Flow and Lockup contracts and before `setNativeToken()` is called, then their funds will be withdrawn as fees whenever `transferFeesToComptroller()` is called.

Impact Explanation: Medium, because user funds of only the streams created with the native token would be withdrawn as fees. Those streams would be defunct until the correct balance of those streams is returned to the contract at which time the sender can cancel the streams to ensure sender and recipient each receive the correct balance.

Likelihood Explanation: Low, because this only can happen when the native token is not set in a timely manner.

Recommendation: Consider setting the native token in the constructor of `ComptrollerManager` (by querying the Comptroller) or ensure the deployment script sets the native token in the same transaction that the contract is deployed. Additionally, consider letting the comptroller cancel streams that incidentally have been created with the native token so funds are distributed correctly.

Sablier: We will ensure that native token is set at the time of the deployment.

Cantina Managed: Acknowledged.

3.4 Gas Optimization

3.4.1 `unlockAmount` calculation can be delayed

Severity: Gas Optimization

Context: SablierMerkleVCA.sol#L221-L243

Description: `_calculateClaimAmount()` calculates the amount that can be claimed at different times. For `MerkleVCA`, which encourages later claims, we can assume that most of the claimants will claim after `VESTING_END_TIME` for maximizing their claimed amounts. Therefore the calculation of `unlockAmount = ud(fullAmount).mul(UNLOCK_PERCENTAGE).intoUint128()` can be delayed and moved into the `else` block where it is required instead of calculating it even for the `claimTime >= VESTING_END_TIME` condition where it is not required.

Recommendation: Consider moving `unlockAmount = ud(fullAmount).mul(UNLOCK_PERCENTAGE).intoUint128()` into the `else` block.

Sablier: Fixed in PR 175.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Malicious/compromised airdrop admins can abuse clawback functionality causing loss of funds to eligible participants

Severity: Informational

Context: SablierMerkleBase.sol#L130-L145, SablierMerkleBase.sol#L206-L210

Summary: Malicious/compromised airdrop admins can abuse clawback functionality to withdraw unclaimed tokens causing loss of funds to eligible participants.

Finding Description: `SablierMerkleBase` provides a `clawback()` function that can be called by airdrop admins to withdraw some/all unclaimed tokens. The motivation for this feature is to allow flexibility to airdrop admins for recovering from any accidental errors in the airdrop campaign configuration such as incorrect merkle root or overfunding. Clawback is allowed after campaign expiry or before expiry if either no one has claimed yet or at least one has claimed but the grace period of 7 days has not passed since the first claim.

However, this flexibility comes with a tradeoff that campaign recipients risk getting rug-pulled by the campaign admin during the clawback period.

Impact Explanation: High, because clawback may cause full/partial loss of funds to eligible recipients depending on when it was triggered, how much amount was withdrawn and whether the campaign admin compensated for the loss in any other way (e.g. another airdrop). If clawback happens before anyone claims or after expiry then all eligible recipients are equally affected or lost their opportunity. However, if the clawback happens during the grace period of 7 days, then it may unfairly affect only the unclaimed.

Likelihood Explanation: Very low, because a malicious clawback will be apparent to everyone (event `Clawback` logged) and negatively affects the reputation of airdrop project.

Recommendation: Consider:

1. Weighing the tradeoff of clawback flexibility to airdrop admins versus rug-ability of airdrop participants.
2. Guaranteeing a locked `aggregateAmount` at campaign start that can be verified onchain by participants.
3. Providing tools for airdrop admins to validate their campaign configuration beforehand.
4. Preventing clawbacks during campaign without a grace period, especially if there's already been a successful claim.
5. Making it configurable (instead of hardcoded 7 days) and notifying participants about it if grace period is a must-have for flexibility.

Sablier: Airdrop campaign is an interaction between the campaign owners and the recipients and clawback is designed to offer flexibility to the campaign owners regardless of their intention.

Cantina Managed: Acknowledged.

3.5.2 Malicious airdrop campaign creator can grief claimants by insufficiently funding campaign

Severity: Informational

Context: SablierFactoryMerkleInstant.sol#L43-L74

Summary: Malicious airdrop campaign creator can grief claimants by insufficiently funding campaign so that some/all of their claims fail when attempted.

Finding Description: While the airdrop campaign creation factories accept an `aggregateAmount` parameter from the creator to log "*The total amount of ERC-20 tokens to be distributed to all recipients*", the flow does not enforce the creator to fund the campaign in the context of the protocol. This is apparently to provide creator the flexibility to fund the campaign out-of-band later and even from a different account.

However, there is no functionality for airdrop participants to easily verify if the campaign has been sufficiently funded so that they can be assured of their claims being successful (assuming their claims were correctly included in the Merkle tree).

Impact Explanation: High, because malicious airdrop campaign creators can grief claimants by insufficiently funding campaign and denying them their airdrop funds after creating the campaign and advertising it with the logged events. If creators partially fund it then later claimants will unfairly be denied airdrops.

Likelihood Explanation: Very low, because this affects the airdrop project reputation and the creator could have chosen to not perform the airdrop itself instead of griefing.

Recommendation: Consider providing an option for creator to deposit funds within the context of the protocol so that the actually funded amount is logged for airdrop participants to claim without the risk of being griefed because of insufficient funds.

Sablier: Allowing campaign creators to fund the campaigns partially is an intentional design choice since airdrops frequently involve substantial amounts of funds and therefore we allow creators to fund their campaigns in a way that best suits their needs.

Cantina Managed: Acknowledged.

3.5.3 Comptroller's ability to only lower the fee can be circumvented by setting a malicious oracle

Severity: Informational

Context: SablierMerkleBase.sol#L148-L166, SablierComptroller.sol#L338-L346

Summary: The comptroller is only allowed to lower the fee, but by maliciously setting an oracle that returns a lower price for the native token it could still increase the required minimum fee.

Finding Description: To provide trust to the airdrop projects, once the airdrop is created, the protocol only allows the comptroller to lower the fee expressed in USD, not increase it.

However when claiming, the comptroller is queried to convert the fee amount from USD to the chain's native token through an oracle set in the comptroller. The comptroller is authorized to set the oracle used for this conversion. This indirectly allows the comptroller to set an oracle that increases the fee amount expressed in native token.

Impact Explanation: High, because technically the fee could be set so high that fee imposed is unreasonably high compared to the user's claim. In the case of airdrops without expiration the funds could be economically locked in the contract.

Likelihood Explanation: Very low, because this would require malicious intent from the Sablier team which would harm their reputation.

Recommendation: The possible reputation damage may be enough for projects to not need any other mechanism to safeguard from malicious intent. As being able to change the oracle is also required to safeguard from oracle failure, a balance must be struck between providing trust between both parties and fault tolerance of the protocol.

Sablier: While it's technically possible, it's highly unlikely since these functions are only called by the admin. We will exercise the maximum caution when changing oracles.

Cantina Managed: Acknowledged.

3.5.4 Missing `ComptrollerManager` convenience function to query required fee

Severity: Informational

Context: SablierMerkleBase.sol#L245-L253, SablierFlow.sol#L950-L956, SablierLockup.sol#L648-L654

Description: Withdrawals in lockup/flow and claims in airdrops require callers to pay a minimum protocol fee as determined by `comptroller.calculateFlowMinFeeWeiFor()`. However, there is no convenience function for users interacting with those contracts to directly query the required fee for successful withdrawals/claims.

Recommendation: Consider adding a `ComptrollerManager` convenience function to query required fee.

Sablier: Fixed in PR 1270, PR 454 and PR 172.

Cantina Managed: Reviewed that PR 1270, PR 454 and PR 172 add a convenience `calculateMinFeeWei()` function to lockup, flow and airdrop.

3.5.5 Returning zero in `calculateForgoneAmount()` when `claimTime < VESTING_START_TIME` may be unexpected

Severity: Informational

Context: SablierMerkleVCA.sol#L130-L143

Description: `calculateForgoneAmount()` returns the amount of `fullAmount` that will be forgone/forfeited if the `MerkleVCA` airdrop is claimed at `claimTime`. If `claimTime < VESTING_START_TIME`, this returns zero with the justification that no amount will be forgone because no amount can be claimed at such a time.

However, this may be confusing/unexpected and may lead to forfeited funds for a user who checks this with sentinel value before `VESTING_START_TIME` and is told that they will forfeit nothing (zero forgone amount) if they claim now but will forfeit tokens if they claim later, for e.g., in the next block when `block.timestamp == VESTING_START_TIME`.

Recommendation: Consider reverting for this scenario instead of returning zero.

Sablier: Fixed in PR 174.

Cantina Managed: Fix verified.

3.5.6 Forcing `MerkleVCA` campaigns to expire only after a week of `vestingEndTime` may be too restrained

Severity: Informational

Context: SablierMerkleVCA.sol#L95-L101

Description: `MerkleVCA` campaigns are allowed to expire only after a week or more of `vestingEndTime`. This is to allow all airdrop participants sufficient time to claim in that period before allowing the campaign creator the ability to clawback airdrop funds.

While this enforcement of a claim period is helpful, the hardcoded minimum of one week may be too restrained for certain airdrop campaigns depending on the campaign goals and expected use of airdropped tokens.

Recommendation: Consider making the minimum time from `vestingEndTime` to expiration as a campaign configurable option instead of enforcing it in protocol to be a week for all campaigns.

Sablier: We believe hardcoding 1 week here is acceptable. Since it represents the minimum time difference between the vesting end time and the expiry, users have the flexibility to choose an expiry period ranging from one week after the end time to an indefinite duration.

Cantina Managed: Acknowledged.